Fast variational quantum algorithms for training neural networks and solving convex optimizations

Changpeng Shao*
Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

(Received 19 February 2019; revised manuscript received 13 March 2019; published 17 April 2019)

Variational hybrid quantum classical algorithms to optimizations are important applications for near-term quantum computing. This paper proposes two quantum algorithms (the second one is variational) for training neural networks. Both of them obtain exponential speedup at the number of samples and polynomial speedup at the dimension of the samples over classical training algorithms. Moreover, the proposed quantum algorithms return the classical information of the training weight so that the outputs can be used directly to solve other problems. For practicality, we draw the quantum circuits to implement the two algorithms. Finally, as an inspiration, we show how to apply the variational algorithm to achieve speedup at the number of constraints in solving convex optimization problems.

DOI: 10.1103/PhysRevA.99.042325

I. INTRODUCTION

A neural network is an important computing system of machine learning that is inspired by the biological neural networks [1,2]. A commonly used neural network is feed forward, in which the neurons are organized by layers: an input layer, several hidden layers, and an output layer. A neural network with many hidden layers is complicated to train by a backpropagation algorithm. However, when the hidden layer contains too many neurons, it is reasonable to use multiple hidden layers to reduce the number of neurons in each layer. In deep learning, the network usually contains at least ten hidden layers. Because of many hidden layers and many weights, training a deep network is time consuming. However, it turns out that deep learning has become more and more important in many fields nowadays, such as automatic speech recognition, image recognition, natural language processing, and so on [3–5].

Supervised learning is a commonly used approach to train neural networks. For instance, let $(\mathbf{x}^{(1)}, r^{(1)}), \ldots, (\mathbf{x}^{(m)}, r^{(m)})$ be m samples, where $\mathbf{x}^{(t)} \in \mathbb{R}^d$ and $r^{(t)}$ is the label of $\mathbf{x}^{(t)}$. To train a perceptron, we can apply the gradient descent method to minimize the following cost function:

$$\min_{\mathbf{w}} E = \frac{1}{2m} \sum_{t=1}^{m} [\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}) - r^{(t)}]^2, \tag{1}$$

where φ is the activation function, such as the sigmoid function, hyperbolic tangent function, and so on.

For deep learning, the performance increases when feeding the network with more and more data. Therefore, we can imagine that m is large in Eq. (1). However, when m is large, the classical training algorithm becomes inefficient since its complexity is at least linear at m.

This paper aims to provide efficient quantum algorithms to improve the efficiency of training neural networks. The main idea is as follows: First, apply a quantum computer to accelerate the evaluation of the gradient of the cost function. Then implement the gradient descent method on a classical computer with the result of the first step.

There are two reasons why we choose to use the above quantum classical hybrid algorithm to train a neural network: (1) Neural networks solve real-work problems; it is more useful to obtain classical information of $\mathbf{w}$. A quantum computer usually returns the quantum state of $\mathbf{w}$. If d is the size of $\mathbf{w}$, then it costs at least $O(d)$ to read out all the entries of $\mathbf{w}$ from its quantum state. Therefore, it seems unlikely to achieve high speedup at d in a quantum computer. (2) For (deep) neural networks, the number of neurons in each layer is not large, that is, $d = O(1)$. Thus it is not significant to achieve speedup at d.

As for the evaluation of the gradient in the first step, we will propose two quantum algorithms (in Secs. III and IV, respectively) that can achieve exponential speedup at m over classical algorithms. More precisely, in each step of the iteration, the complexity of the quantum training algorithm is $O[d^{1.5}(\log_2 dm)/\epsilon^2]$, where ϵ is the accuracy. However, the complexity of the classical training algorithm is $O(d^2 m)$.

The first quantum algorithm computes the gradient with respect to the weight $\mathbf{w} = (w_0, \ldots, w_{d-1})$. Besides the influence of the samples (this is acceptable), the efficiency of this quantum algorithm is affected by the norm of the weight, which can be large. The second quantum algorithm overcomes this drawback by considering a new representation of the weight introduced in Sec. II. More precisely, in Sec. II, we will show that for any d-dimensional real vector $\mathbf{w}$, there exist $d-1$ angle parameters $\vec{\theta} = (\theta_1, \ldots, \theta_{d-1})$ and a unitary operator $U(\vec{\theta})$, generated by two-dimensional rotations, such that $|\mathbf{w}\rangle = U(\vec{\theta})|0\rangle^{\otimes \log_2 d}$. Because of this, the second quantum algorithm trains $\vec{\theta}$ instead of $w_0, \ldots, w_{d-1}$. Since $\vec{\theta}$ appears as angles of rotations, it will not change the norm of the weight during the training procedure. Thus, if the initial weight has the unit norm, then the training weight always has the unit

*cpshao@amss.ac.cn

norm. Therefore, the second algorithm is not affected by the norm of the weight.

Besides designing a quantum algorithm, one advantage of the representation $|\mathbf{w}\rangle = U(\vec{\theta})|0\rangle^{\otimes \log_2 d}$ is the calculation of the inner product $\langle \mathbf{x}^{(t)}|\mathbf{w}\rangle$. It provides an efficient quantum circuit to calculate $\langle \mathbf{x}^{(t)}|\mathbf{w}\rangle = \langle \mathbf{x}^{(t)}|U(\vec{\theta})|0\rangle^{\otimes \log_2 d}$. Another advantage is that the derivative of $U(\vec{\theta})$ with respect to θ_i is a linear combinations of two unitaries, which can be implemented in the same circuit as $U(\vec{\theta})$. Thus, similar to the estimation of $\langle \mathbf{x}^{(t)}|\mathbf{w}\rangle$, the formula $\frac{\partial}{\partial\theta_i}\langle \mathbf{x}^{(t)}|\mathbf{w}\rangle = \langle \mathbf{x}^{(t)}|\frac{\partial}{\partial\theta_i}U(\vec{\theta})|0\rangle^{\otimes \log_2 d}$ also offers an easy method to calculate the gradient of the cost function. We will discuss these in detail in Sec. IV.

Actually, the above idea to construct the second algorithm is known as a variational algorithm [6]. It is an important approach to solve useful problems (such as machine learning [7]) in near-future quantum computing devices with a limited number of elementary gates and qubits. The unitary operators used in variational algorithms continuously depend on some parameters, such as $U(\vec{\theta})$ constructed in this paper. One great advantage of such unitaries is that its derivative can be implemented in the same quantum circuit as $U(\vec{\theta})$. A simple modification at the parameters gives the quantum circuit to calculate the derivative of $U(\vec{\theta})$. Many works have been recently reported about variational algorithms, such as combinatorial optimization problems [8,9], variational quantum eigensolvers [10,11], quantum neural network [12–14], quantum classifier [15–17], etc. This paper can be viewed as another important application in this direction.

Finally, as an extension of the quantum algorithm to train a neural network, we will study quantum algorithms to solve certain types of convex optimization problems (e.g., linear programming, quadratic programming, and geometric programming) in Sec. VI. Based on the barrier method—a particular interior-point method—many programming problems can be changed into a similar form as the optimization problem (1). Therefore, by some simple modifications, a quantum algorithm to train a neural network is also applicable to solve them with certain speedup at the number of constraints.

II. REPRESENTATION OF REAL VECTORS

Let $\mathbf{w} = (w_0, \ldots, w_{d-1}) \in \mathbb{R}^d$ be a real vector. For convenience, we assume that $d = 2^n$ for some n; otherwise, we can intentionally insert some zeros into $\mathbf{w}$.

For any θ, define

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

as the rotation with angle θ in the plane spanned by $|0\rangle, |1\rangle$. In the following, we give a method to prepare the quantum state $|\mathbf{w}\rangle = \frac{1}{\|\mathbf{w}\|_2}\sum_i w_i|i\rangle$ of $\mathbf{w}$. For convenience, assume that $\|\mathbf{w}\|_2 = 1$, which is a reasonable assumption in preparing a quantum state. However, the following analysis also holds if $\|\mathbf{w}\|_2 \neq 1$.

For coherence, set $\mathbf{w}^{(0)} = \mathbf{w}$ and $w_i^{(0)} = w_i$ for all i. For $0 \leqslant i \leqslant 2^{n-1} - 1$, denote $w_i^{(1)} = \sqrt{(w_{2i}^{(0)})^2 + (w_{2i+1}^{(0)})^2}$ and $\theta_i^{(1)}$ as the angle satisfies

$$\cos\left(\theta_i^{(1)}\right) = \frac{w_{2i}^{(0)}}{w_i^{(1)}}, \qquad \sin\left(\theta_i^{(1)}\right) = \frac{w_{2i+1}^{(0)}}{w_i^{(1)}}.$$

Then it is easy to verify that

$$|\mathbf{w}^{(0)}\rangle = \sum_{i=0}^{2^{n-1}-1} w_i^{(1)}|i\rangle\left[\cos\left(\theta_i^{(1)}\right)|0\rangle + \sin\left(\theta_i^{(1)}\right)|1\rangle\right]. \quad (2)$$

Set

$$|\mathbf{w}^{(1)}\rangle = \sum_{i=0}^{2^{n-1}-1} w_i^{(1)}|i\rangle. \quad (3)$$

Then, Eq. (2) is equivalent to

$$|\mathbf{w}^{(0)}\rangle = \left(\sum_{i=0}^{2^{n-1}-1} |i\rangle\langle i| \otimes R\left(\theta_i^{(1)}\right)\right)|\mathbf{w}^{(1)}\rangle|0\rangle. \quad (4)$$

Similarly, for $|\mathbf{w}^{(1)}\rangle$, we can find $|\mathbf{w}^{(2)}\rangle$ and $\theta_i^{(2)}$ (where $i = 0, \ldots, 2^{n-2} - 1$), such that

$$|\mathbf{w}^{(1)}\rangle = \left(\sum_{i=0}^{2^{n-2}-1} |i\rangle\langle i| \otimes R\left(\theta_i^{(2)}\right)\right)|\mathbf{w}^{(2)}\rangle|0\rangle. \quad (5)$$

Finally, it is easy to prove the following result by continuing the above procedure in n steps.

Theorem 1. Let $\mathbf{w}$ be a 2^n-dimensional real vector; then there exist $2^n - 1$ angle parameters $\vec{\theta}^{(i)} = (\theta_0^{(i)}, \ldots, \theta_{2^{n-i}-1}^{(i)})$, where $i = 1, \ldots, n$, such that

$$|\mathbf{w}\rangle = U_1(\vec{\theta}^{(1)})U_2(\vec{\theta}^{(2)})\cdots U_n(\vec{\theta}^{(n)})|0\rangle^{\otimes n}. \quad (6)$$

Moreover,

$$U_i(\vec{\theta}^{(i)}) = \sum_{j=0}^{2^{n-i}-1} |j\rangle\langle j| \otimes R\left(\theta_j^{(i)}\right) \otimes I_{2^{i-1}}. \quad (7)$$

In the following, we will simply write $\vec{\theta} = (\vec{\theta}^{(1)}, \ldots, \vec{\theta}^{(n)})$ and $U(\vec{\theta}) = U_1(\vec{\theta}^{(1)})U_2(\vec{\theta}^{(2)})\cdots U_n(\vec{\theta}^{(n)})$.

For instance, suppose that $\mathbf{w} = (w_0, w_1, w_2, w_3)$. Then, $\theta_0^{(1)}$, $\theta_1^{(1)}$, and $\theta_0^{(2)}$ satisfy

$$\cos\theta_0^{(1)} = \frac{w_0}{\sqrt{w_0^2 + w_1^2}}, \quad \sin\theta_0^{(1)} = \frac{w_1}{\sqrt{w_0^2 + w_1^2}},$$

$$\cos\theta_1^{(1)} = \frac{w_2}{\sqrt{w_2^2 + w_3^2}}, \quad \sin\theta_1^{(1)} = \frac{w_3}{\sqrt{w_2^2 + w_3^2}},$$

$$\cos\theta_0^{(2)} = \frac{\sqrt{w_0^2 + w_1^2}}{\|\mathbf{w}\|_2}, \quad \sin\theta_0^{(2)} = \frac{\sqrt{w_2^2 + w_3^2}}{\|\mathbf{w}\|_2}.$$

To prepare the quantum state of $\mathbf{w}$, we first apply $R(\theta_0^{(2)}) \otimes I$ to $|0\rangle|0\rangle$ to get

$$\frac{\sqrt{w_0^2 + w_1^2}}{\|\mathbf{w}\|_2}|0\rangle|0\rangle + \frac{\sqrt{w_2^2 + w_3^2}}{\|\mathbf{w}\|_2}|1\rangle|0\rangle.$$

In the above quantum state, if the first qubit is $|0\rangle$, then apply $R(\theta_0^{(1)})$ to the second register; if the first qubit is $|1\rangle$, then apply

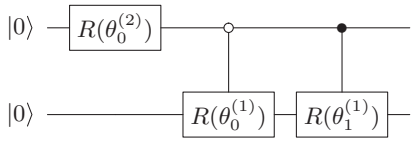

FIG. 1. Quantum circuit to prepare $|\mathbf{w}\rangle$.

$R(\theta_1^{(1)})$ to the second register. By doing so, we obtain

$$\frac{1}{\|\mathbf{w}\|_2}(w_0|00\rangle + w_1|01\rangle + w_2|10\rangle + w_3|11\rangle).$$

This gives the quantum state of $\mathbf{w}$.

Figure 1 depicts the quantum circuit of preparing $|\mathbf{w}\rangle$. The circuit contains three elementary gates. In general, for a d-dimensional real vector, the circuit contains $d-1$ elementary gates.

The above idea is similar to the binary tree structure proposed in [18] or qRAM proposed in [19] to storing vectors in a quantum computer (see Fig. 2, for example). With this data structure, one can prepare a quantum state in polynomial logarithm time.

In 2002, Grover and Rudolph [20] proposed a quantum algorithm to prepare a quantum state with the assumption that we know the distribution of the vector $\mathbf{w}$. In other words, this assumption is equivalent to assume that we know $\vec{\theta}$ of $\mathbf{w}$. Actually, building the tree structure is equivalent to obtain $\vec{\theta}$ of $\mathbf{w}$. Both ideas of preparing the quantum state are close to each other. However, by Eqs. (6) and (7), one can figure out that the gate complexity to prepare the quantum state of $\mathbf{w}$ is still $O(d)$ even though we know $\vec{\theta}$ (see Fig. 1 for the case $d=4$).

However, if we have an efficient oracle $\mathcal{O}$ to query all the angles, then we can prepare the quantum state of $\mathbf{w}$ by using $O[\text{poly}\log_2(d)]$ elementary gates. The oracle $\mathcal{O}$ here is defined as

$$\mathcal{O}: |i,j\rangle|0\rangle \to |i,j\rangle\left|\theta_j^{(i)}\right\rangle. \tag{8}$$

To see this, assume that we already have $|\mathbf{w}^{(1)}\rangle$ [see Eq. (3) for definition] in polylog time; then we use the oracle to prepare $\sum_i w_i^{(1)}|i\rangle|\theta_i^{(1)}\rangle$. Thus, we can prepare $\sum_i w_i^{(1)}|i\rangle|\theta_i^{(1)}\rangle \otimes R(\theta_i^{(1)})|0\rangle$ by viewing $|\theta_i^{(1)}\rangle$ as a control register. Finally, apply the inverse of the oracle to eliminate $|\theta_i^{(1)}\rangle$ to get $|\mathbf{w}^{(0)}\rangle = \sum_i w_i^{(1)}|i\rangle \otimes R(\theta_i^{(1)})|0\rangle$.

With the help of the oracle, we can implement the above procedure in a more efficient way than using $U_1(\vec{\theta^{(1)}})$ directly to prepare $|\mathbf{w}^{(0)}\rangle$. The main ingredient is the second step of applying the control rotation. For any j, assume that $\theta_j^{(1)} \approx \alpha_j^{(1)} 2\pi$, where $0 \leqslant \alpha_j^{(1)} \leqslant 1$ and $\alpha_j^{(1)} \approx$

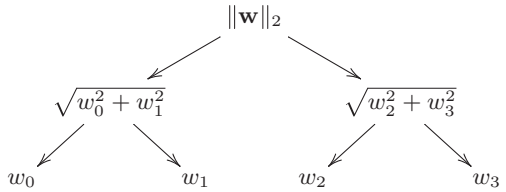

FIG. 2. Binary tree data structure.

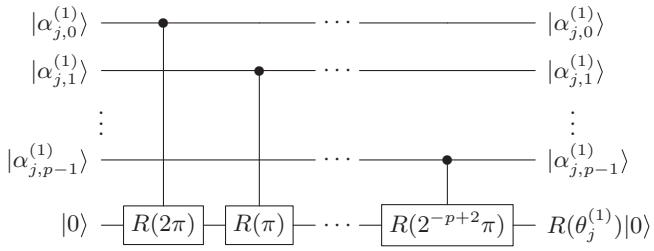

FIG. 3. Circuit implementation of $R(\theta_j^{(1)})$.

$\sum_{k=0}^{p-1}\alpha_{j,k}^{(1)}2^{-k}$ in binary form. Then, we have $R(\theta_j^{(1)}) \approx \prod_{k=0}^{p-1} R(\alpha_{j,k}^{(1)}2^{-k+1}\pi)$. It can be implemented in the following circuit (see Fig. 3), which is independent of the input $\theta_j^{(1)}$. It only depends on the oracle and the precision p. Therefore, we have a simple uniform circuit to prepare $|\mathbf{w}\rangle = |\mathbf{w}^{(0)}\rangle$ from $|\mathbf{w}^{(1)}\rangle$. By induction, we can obtain $|\mathbf{w}^{(1)}\rangle$ from $|\mathbf{w}^{(2)}\rangle$ in polylog time, and so on. As a result, we can prepare $|\mathbf{w}^{(0)}\rangle$ from $|0\rangle^{\otimes \log_2 d}$ in polylog time.

Assume that the oracle (8) exists for the training samples $\{\mathbf{x}^{(t)}: t=1,\ldots,m\}$. Let $\sum_t \alpha_t|t\rangle$ be any efficiently prepared quantum state; then we can prepare $\sum_t \alpha_t|t\rangle|\mathbf{x}^{(t)}\rangle$ efficiently by control operation in a similar way as to prepare $|\mathbf{w}\rangle$. In this case, the oracle can be extended into

$$\widetilde{\mathcal{O}}: |t,i,j\rangle|0\rangle \to |t,i,j\rangle\left|(\theta^{(t)})_j^{(i)}\right\rangle, \tag{9}$$

where $(\theta^{(t)})_j^{(i)}$ are the angle parameters of $\mathbf{x}^{(t)}$.

III. FAST QUANTUM ALGORITHM TO TRAIN NEURAL NETWORKS

In this section, we will establish a fast quantum algorithm to train feed-forward neural networks. To achieve this goal, we start from the simplest neural network: a perceptron. However, the idea is also true for training the general neural network, which we will discuss at the end of this section. To train a perceptron, we need to solve the minimization problem (1).

In general, we should use an iteration method, such as the gradient descent method or Newton's method, to solve the minimization problem (1). Simple calculation shows that the gradient of E is

$$\mathbf{g}(\mathbf{w}) = \frac{1}{m}\sum_{t=1}^{m}\psi(\mathbf{x}^{(t)}\cdot\mathbf{w})\mathbf{x}^{(t)}, \tag{10}$$

where

$$\psi(\mathbf{x}^{(t)}\cdot\mathbf{w}) = [\varphi(\mathbf{x}^{(t)}\cdot\mathbf{w}) - r^{(t)}]\varphi'(\mathbf{x}^{(t)}\cdot\mathbf{w}). \tag{11}$$

Then the updating rule of the gradient descent method is

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \frac{\eta}{m}\sum_{t=1}^{m}\psi(\mathbf{x}^{(t)}\cdot\mathbf{w}(n))\mathbf{x}^{(t)}, \tag{12}$$

where $\eta \in (0,1]$ is the learning factor and n is the number of iteration.

For a classical computer, the main difficulty to implement the updating rule (12) is the evaluation of the summation,

$$\mathbf{g}_i(\mathbf{w}(n)) = \frac{1}{m}\sum_{t=1}^{m}\psi(\mathbf{x}^{(t)}\cdot\mathbf{w}(n))\mathbf{x}_i^{(t)},$$

for $i = 1, \ldots, d$, where $\mathbf{x}_i^{(t)}$ is the ith entry of vector $\mathbf{x}^{(t)}$. Next, we show how to achieve speedup at the evaluation of $\mathbf{g}_i(\mathbf{w}(n))$ in a quantum computer. For simplicity, we first consider the case that $\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))\mathbf{x}_i^{(t)} \geqslant 0$ for all t; then we extend it into the general case.

We assume that the quantum state of samples is already prepared by the algorithm proposed in Sec. II. To make the algorithm efficient, we assume that the oracle (9) exists for the samples. We also assume that the norms of the samples are given in advance. As for the weight, since it updates in each step of the iteration, we can compute all the angles defined in Theorem I classically. This requires $O(d)$ operations, which is acceptable to us since we do not intend to achieve speedup at the dimension d. Thus, with $O(d)$ operations, we can enlarge the oracle $\widetilde{\mathcal{O}}$ to include the angle parameters of $\mathbf{w}(n)$. Therefore, the quantum state of $\mathbf{w}(n)$ can be prepared efficiently in the same way as preparing the states of samples when having this extended oracle.

In the following, we propose the quantum algorithm to approximate the gradient $\mathbf{g}_i(\mathbf{w}(n))$ for any fixed $i \in \{1, \ldots, d\}$. The algorithm depends on the amplitude estimation algorithm [21], which is briefly reviewed in the Appendix.

Algorithm 1. Fast quantum algorithm to estimate the gradient $\mathbf{g}_i(\mathbf{w}(n))$.

Step 1. Prepare $|\phi_1\rangle = \frac{1}{\sqrt{m}} \sum_{t=1}^{m} |t\rangle$.

Step 2. For any $t = 1, \ldots, m$, denote

$$|\phi_{t,n}\rangle = \frac{1}{\sqrt{2}}[|\mathbf{x}^{(t)}\rangle|+\rangle + |\mathbf{w}(n)\rangle|-\rangle].$$

By viewing $|t\rangle$ as a control qubit in $|\phi_1\rangle$, we can prepare the following state:

$$|\phi_2\rangle = \frac{1}{\sqrt{m}} \sum_{t=1}^{m} |t\rangle \otimes |\phi_{t,n}\rangle.$$

Step 3. Apply amplitude estimation [21] to estimate the amplitude of $|1\rangle$ of the second register of $|\phi_{t,n}\rangle$; then, we obtain an ϵ-approximate $q_{t,n}$ of $\mathbf{x}^{(t)} \cdot \mathbf{w}(n)$. Store $q_{t,n}$ in an ancilla register [see Eq. (A5)],

$$|\phi_3\rangle = \frac{1}{\sqrt{m}} \sum_{t=1}^{m} |t\rangle \otimes |\phi_{t,n}\rangle \otimes |q_{t,n}\rangle.$$

Step 4. Set $L = 1/\max_t |\psi(q_{t,n})\mathbf{x}_i^{(t)}|$ and $f_i^{(t)}(s) = \psi(s)\mathbf{x}_i^{(t)}L$; then, apply $I \otimes I \otimes U_{f_i^{(t)}}$ to $|\phi_3\rangle$ to prepare

$$|\phi_4\rangle = \frac{1}{\sqrt{m}} \sum_{t=1}^{m} |t\rangle \otimes |\phi_{t,n}\rangle \otimes |q_{t,n}\rangle \otimes \left|\psi(q_{t,n})\mathbf{x}_i^{(t)}L\right\rangle.$$

Step 5. Apply control rotation to $|\phi_4\rangle$ to get

$$|\phi_5\rangle = \frac{1}{\sqrt{m}} \sum_{t=1}^{m} |t\rangle \otimes |\phi_{t,n}\rangle \otimes |q_{t,n}\rangle \otimes \left|\psi(q_{t,n})\mathbf{x}_i^{(t)}L\right\rangle \otimes \left[\sqrt{\psi(q_{t,n})\mathbf{x}_i^{(t)}L}|0\rangle + \sqrt{1 - \left|\psi(q_{t,n})\mathbf{x}_i^{(t)}\right|L}|1\rangle\right]. \tag{13}$$

Step 6. Undo steps 2–4 to get

$$|\phi_6\rangle = \frac{\sqrt{L}}{\sqrt{m}} \sum_{t=1}^{m} \sqrt{\psi(q_{t,n})\mathbf{x}_i^{(t)}}|t\rangle|0\rangle + (\cdots)|1\rangle.$$

Step 7. Apply the amplitude estimation to estimate the amplitude of $|0\rangle$ of the second register of $|\phi_6\rangle$.

In step 2, since the quantum states of $\mathbf{x}^{(t)}$ and $\mathbf{w}(n)$ are prepared in an efficient way discussed at the end of Sec. II, we can make sure that the complexity of the control operation to prepare $|\phi_2\rangle$ is independent of m.

In steps 3 and 7, the amplitude estimation is performed for different purposes. In step 3, we still need the information of $q_{t,n}$ to do further operations, and thus we store it in an ancilla register as Eq. (A5) did. That is, we perform no measurement in the amplitude estimation in step 3. Similar to Eq. (A5), the amplitude estimation can be performed in parallel by control operations. However, in step 7, we need an output about the estimate of $\mathbf{g}_i(\mathbf{w}(n))$ to update the weight by Eq. (12). Hence, in this step, we need to perform measurements at the amplitude estimation.

In step 3, the amplitude of $|1\rangle$ of the second register of $|\phi_{t,n}\rangle$ is $\sqrt{[1 - \langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle]/2}$. Using the notation in the amplitude estimation algorithm in the Appendix, the amplitude corresponds to $\cos\theta_{t,n}$ for some angle $\theta_{t,n}$. The amplitude estimation returns an approximate $\tilde{\theta}_{t,n}$ of $\theta_{t,n}$ or $-\theta_{t,n}$. Since the cosine function is even, we can obtain an approximate of $\sqrt{[1 - \langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle]/2}$ by computing $\cos\tilde{\theta}_{t,n}$. Thus, $1 - 2\cos^2\tilde{\theta}_{t,n}$ gives an approximate of $\langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle$, and $q_{t,n} = \|\mathbf{x}^t\|_2\|\mathbf{w}(n)\|_2(1 - 2\cos^2\tilde{\theta}_{t,n})$ gives an approximate of $\mathbf{x}^{(t)} \cdot \mathbf{w}(n)$. Therefore, the function discussed in Eq. (A5) is $f(s) = \|\mathbf{x}^t\|_2\|\mathbf{w}(n)\|_2(1 - 2s^2)$. It is not hard to show that U_f is efficient when $\|\mathbf{x}^t\|_2\|\mathbf{w}(n)\|_2$ are known [22]. Therefore, we can store $q_{t,n}$ into the ancilla register efficiently. And the complexity is independent of m.

In step 4, a proper upper bound of $\max_t |\psi(q_{t,n})\mathbf{x}_i^{(t)}|$ is also enough to determine the value L. We will discuss this later in this section in that it relates to other notations.

In step 5, the control rotation is efficient since we can construct a uniform circuit to implement these control rotations. See Fig. 5 in Sec. V.

Theorem 2. Assume that the oracle (9) exists for the training samples. Then, Algorithm 1 returns an ϵ-approximate of $\mathbf{g}_i(\mathbf{w}(n))$ in time,

$$O\big\{(\log_2 dm)(\max_t \|\mathbf{x}^{(t)}\|_2) \times \big[\max_t \left|\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))\mathbf{x}_i^{(t)}\right|\big]\|\mathbf{w}(n)\|_2/\epsilon^2\big\}. \tag{14}$$

Proof. With the analysis above Theorem 2, it suffices to estimate the complexity of steps 3 and 7.

In step 3, the amplitude of $|1\rangle$ of the second register of $|\phi_{t,n}\rangle$ is $\sqrt{[1 - \langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle]/2}$. By amplitude estimation, we obtain $p_{t,n}$ in time $O((\log_2 dm)/\epsilon')$ such that

$$\left|\tfrac{1}{2}[1 - \langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle] - p_{t,n}\right| \leqslant \epsilon'.$$

Thus, $1 - 2p_{t,n}$ gives a $2\epsilon'$-approximate of $\langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle$, and $\|\mathbf{x}^{(t)}\|_2\|\mathbf{w}(n)\|_2(1 - 2p_{t,n})$ gives a $2\epsilon'\|\mathbf{x}^{(t)}\|_2\|\mathbf{w}(n)\|_2$-approximate of $\mathbf{x}^{(t)} \cdot \mathbf{w}(n)$. To make this error small in size ϵ, we set $\epsilon = 2\epsilon'\|\mathbf{x}^{(t)}\|_2\|\mathbf{w}(n)\|_2$. As a result, we can get an ϵ-approximate of $\mathbf{x}^{(t)} \cdot \mathbf{w}(n)$ in time $O((\log_2 dm)\|\mathbf{x}^{(t)}\|_2\|\mathbf{w}(n)\|_2/\epsilon)$.

In step 7, by amplitude estimation, we get $g_{t,n}$ in time $O((\log_2 dm)(\max_t \|\mathbf{x}^{(t)}\|_2)\|\mathbf{w}(n)\|_2/\epsilon\epsilon'')$ such that

$$\left| g_{t,n} - \frac{L}{m}\sum_{t=1}^{m} \psi(q_{t,n})\mathbf{x}_i^{(t)} \right| \leqslant \epsilon''.$$

Thus, $g_{t,n}/L$ returns an $\frac{\epsilon''}{L}$-approximate of $\mathbf{g}_i(\mathbf{w}(n))$. By setting $\epsilon'' = \epsilon L$, we can get an ϵ-approximate of $\mathbf{g}_i(\mathbf{w}(n))$ in time (14). ■

Note that when closing to the end of the training, the gradient can be very small. To obtain a good approximate of the gradient, one idea is to increase the precision ϵ, which in turn increases the complexity (14) of Algorithm 1. Another idea is that we set a threshold ϵ' about the gradient. When the gradient is smaller than or close to ϵ', then we stop the training. This is reasonable since we cannot make sure the gradient is precisely equal to 0 at the end of the training. Moreover, if we choose the precision in Algorithm 1 as ϵ', and if the gradient is smaller than ϵ', then we will obtain an approximate of the gradient to precision ϵ' by Algorithm 1. Hence this approximate is close to ϵ', from which we can believe that the training is close to the end. And we can stop the training now. The complexity has a similar form to (14), except that ϵ is changed into ϵ'.

Remark 1. By a simple modification at step 5, we can also use Algorithm 1 to solve the general case. More precisely, denote $\psi(q_{t,n})\mathbf{x}_i^{(t)} = \psi_{t,n,i}^{+}$ if it is non-negative, and $\psi_{t,n,i}^{-}$ otherwise. Then, using an oracle (or adding an ancilla qubit) to detect the sign of $\psi(q_{t,n})\mathbf{x}_i^{(t)}$, we can prepare

$$\begin{aligned}|\tilde{\phi}_5\rangle = {} & \frac{1}{\sqrt{m}} \sum_{\psi(q_{t,n})\mathbf{x}_i^{(t)} \geqslant 0} |t\rangle \otimes |\phi_{t,n}\rangle \otimes |q_{t,n}\rangle \\ & \otimes \left[\sqrt{\psi_{t,n,i}^{+} L}|0\rangle + \sqrt{1 - \left|\psi(q_{t,n})\mathbf{x}_i^{(t)}|L\right|1\rangle} \right] |0\rangle \\ & + \frac{1}{\sqrt{m}} \sum_{\psi(q_{t,n})\mathbf{x}_i^{(t)} < 0} |t\rangle \otimes |\phi_{t,n}\rangle \otimes |q_{t,n}\rangle \\ & \otimes \left[\sqrt{-\psi_{t,n,i}^{-} L}|0\rangle + \sqrt{1 - \left|\psi(q_{t,n})\mathbf{x}_i^{(t)}|L\right|1\rangle} \right] |1\rangle.\end{aligned}$$

The last qubit $|0\rangle, |1\rangle$ acts as the sign-distinguishing qubit.

By amplitude estimation, we can separately estimate the amplitude of $|0,0\rangle$ and $|0,1\rangle$ of the last two registers in the two summations. The first one gives an approximate of $\mathbf{g}_i(\mathbf{w}(n))^{+} = \frac{1}{m}\sum_{\psi(q_{t,n})\mathbf{x}_i^{(t)} \geqslant 0} \psi_{t,n,i}^{+}$, and the second one gives an approximate of $\mathbf{g}_i(\mathbf{w}(n))^{-} = -\frac{1}{m}\sum_{\psi(q_{t,n})\mathbf{x}_i^{(t)} < 0} \psi_{t,n,i}^{-}$. In consequence, $\mathbf{g}_i(\mathbf{w}(n))^{+} - \mathbf{g}_i(\mathbf{w}(n))^{-}$ returns an approximate of $\mathbf{g}_i(\mathbf{w}(n))$.

Since the gradient contains d components, by Theorem 2, in the nth step of iteration, the complexity to implement the gradient descent method in a quantum computer is

$$\begin{aligned}O\big\{ & d(\log_2 dm)(\max_t \|\mathbf{x}^{(t)}\|_2) \\ & \times \big[\max_{t,i} \left|\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))\mathbf{x}_i^{(t)}\right| \big] \|\mathbf{w}(n)\|_2/\epsilon^2 \big\}. \end{aligned} \tag{15}$$

However, the corresponding complexity for a classical computer is $O(d^2 m)$ [23]. Therefore, a quantum computer achieves an exponential speedup at m and a polynomial speedup at d.

In neural networks, φ is an activation function, which means $\varphi(a) = O(1)$ generally for all $a \in \mathbb{R}$. By Eq. (11), we know that $|\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))| = O(1)$. Therefore, we can simplify (15) into

$$O(d(\log_2 dm)(\max_t \|\mathbf{x}^{(t)}\|_2)(\max_t \|\mathbf{x}^{(t)}\|_\infty)\|\mathbf{w}(n)\|_2/\epsilon^2). \tag{16}$$

In Eq. (16), the complexity also depends on the norm $\|\mathbf{w}(n)\|_2$, which can be large. However, in the next section, we will provide a modified quantum algorithm that can remove the influence of $\|\mathbf{w}(n)\|_2$. Thus, the only influences of the complexity to estimate $\mathbf{g}_i(\mathbf{w}(n))$ in a quantum computer are $\|\mathbf{x}^{(t)}\|_2$ and $\|\mathbf{x}^{(t)}\|_\infty$, or just $\|\mathbf{x}^{(t)}\|_\infty$ if we are not too concerned about the dimension d. In practical applications (such as speech recognition and computer vision), the components of the data are not large, that is, $\|\mathbf{x}^{(t)}\|_\infty = O(1)$. Therefore, we can further simplify (16) into

$$O(d^{1.5}(\log_2 dm)/\epsilon^2). \tag{17}$$

As a result, a quantum learning algorithm performs much better than a classical learning algorithm.

Note that when $|\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))| = O(1)$ and $\|\mathbf{x}^{(t)}\|_\infty = O(1)$, the choice of L in step 4 becomes easier. To be more precise, let A, B be the upper bounds of $|\psi(\mathbf{x}^{(t)} \cdot \mathbf{w}(n))|$ and $\|\mathbf{x}^{(t)}\|_\infty$, respectively; then we can choose $L = 1/AB$. Since we can estimate A, B easily from the activation function and the samples in advance, we can determine L easily too.

At the end of this section, we make some comments about Newton's method and the applications of Algorithm 1 in training deep neural networks and in solving some optimization problems.

(i) In the Newton iteration method, the (i,j)th entry of the Hessian matrix of E equals $\frac{1}{m}\sum_{t=1}^{m}\{[\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}) - r^{(t)}]\varphi''(\mathbf{x}^{(t)} \cdot \mathbf{w}) + [\varphi'(\mathbf{x}^{(t)} \cdot \mathbf{w})]^2\}\mathbf{x}_i^{(t)}\mathbf{x}_j^{(t)}$. By designing a quantum algorithm similar to Algorithm 1, we can also use a quantum computer to speed up the estimation of each entry of the Hessian matrix of E, which in turn speeds up the Newton iteration.

(ii) Algorithm 1 also works for training a neural network with multiple hidden layers. For instance, consider the neural network with one hidden layer and one output. Denote the weights in the hidden layer as $\mathbf{w}^{(j)} = (w_1^{(j)}, \ldots, w_n^{(j)})$, $j = 1, \ldots, l$, and the weight in the output layer as $\mathbf{v} = (v_1, \ldots, v_l)$. Then the cost function is

$$E = \frac{1}{2m}\sum_{t=1}^{m}\left[\varphi\left(\sum_{j=1}^{l}\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}^{(j)})v_j\right) - r^{(t)}\right]^2.$$

It is not hard to show that the gradient vector of E satisfies

$$\begin{aligned}\frac{\partial E}{\partial v_j} = {} & \frac{1}{m}\sum_{t=1}^{m}\left[\varphi\left(\sum_{j=1}^{l}\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}^{(j)})v_j\right) - r^{(t)}\right] \\ & \times \varphi'\left(\sum_{j=1}^{l}\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}^{(j)})v_j\right)\varphi(\mathbf{x}^{(t)} \cdot \mathbf{w}^{(j)}),\end{aligned} \tag{18}$$

and

$$\frac{\partial E}{\partial w_k^{(j)}} = \frac{1}{m}\sum_{t=1}^{m}\left[\varphi\left(\sum_{j=1}^{l}\varphi(\mathbf{x}^{(t)}\cdot\mathbf{w}^{(j)})v_j\right) - r^{(t)}\right] \times\varphi'\left(\sum_{j=1}^{l}\varphi(\mathbf{x}^{(t)}\cdot\mathbf{w}^{(j)})v_j\right)\varphi'(\mathbf{x}^{(t)}\cdot\mathbf{w}^{(j)})v_j\mathbf{x}_k^{(t)}. \tag{19}$$

By performing a swap test [24] (similar to amplitude estimation) in parallel, we can estimate all the inner products in (18) and (19). Then, similar to the construction of Algorithm 1, we can also achieve exponential speedup at m to compute the gradients (18) and (19).

(iii) We can also generalize the idea of Algorithm 1 to improve the solving of more general optimization problems. For instance, let $\mathbf{a}^{(1)},\ldots,\mathbf{a}^{(m)}\in\mathbb{R}^d$ be m real vectors and $f, f_0,\ldots,f_m$ be $(m+2)$ differentiable functions. Consider the following optimization problem:

$$\min_{\mathbf{w}} E = f\left(f_0(\mathbf{w}) + \sum_{i=1}^{m} f_i(\mathbf{a}^{(i)}\cdot\mathbf{w})\right). \tag{20}$$

We can design a fast quantum algorithm similar to Algorithm 1 to compute the gradient of E. Many programming problems are included in the form of (20), such as linear programming, quadratic programming, geometric programming, etc. We will discuss this in more detail in Sec. VI.

IV. IMPROVED QUANTUM ALGORITHM FOR TRAINING NEURAL NETWORKS

In this section, we show how to remove the influence of $\|\mathbf{w}(n)\|_2$ in Algorithm 1. The basic idea is that instead of learning $w_0,\ldots,w_{d-1}$ by the gradient descent method, we learn $\vec{\theta}$. Since $\theta_j^{(i)}$ appears as a rotation angle, updating it will not change the norm of the weight. From this point, it is not hard to figure out how to improve Algorithm 1 by removing the influence of weight. We show some details below.

By definition,

$$\frac{d}{d\theta}R(\theta) = \begin{pmatrix}-\sin\theta & -\cos\theta\\ \cos\theta & -\sin\theta\end{pmatrix} = R\left(\theta+\frac{\pi}{2}\right). \tag{21}$$

Then it is easy to verify that

$$\frac{\partial U_i(\vec{\theta}^{(i)})}{\partial\theta_k^{(i)}} = |k\rangle\langle k|\otimes R\left(\theta_k^{(i)}+\frac{\pi}{2}\right)\otimes I_{2^{i-1}} = \frac{1}{2}\sum_{j=0}^{2^{n-i}-1}|j\rangle\langle j|\otimes R\left(\theta_j^{(i)}+\delta_k^j\frac{\pi}{2}\right)\otimes I_{2^{i-1}} - \frac{1}{2}\sum_{j=0}^{2^{n-i}-1}|j\rangle\langle j|\otimes R\left(\theta_j^{(i)}+\delta_k^j\frac{3\pi}{2}\right)\otimes I_{2^{i-1}}.$$

If we denote $\Delta_{k,1}^{(i)}$, $\Delta_{k,2}^{(i)}$ as the 2^{n-i}-dimensional vector generated by $\delta_k^j\pi/2$ and $3\delta_k^j\pi/2$, respectively, for all $0\leqslant j\leqslant 2^{n-i}-1$, then

$$\frac{\partial U_i(\vec{\theta}^{(i)})}{\partial\theta_k^{(i)}} = \frac{1}{2}\left[U_i\left(\vec{\theta}^{(i)}+\Delta_{k,1}^{(i)}\right) - U_i\left(\vec{\theta}^{(i)}+\Delta_{k,2}^{(i)}\right)\right]. \tag{22}$$

Therefore, we can use the same circuit of $U(\vec{\theta})$ to implement $\partial U(\vec{\theta})/\partial\theta_k^{(i)}$ by modifying the parameters. This is another advantage of the representation discussed in Sec. II. A similar proposal to evaluate the gradient by modifying the parameters has been considered in [25].

Let $|\mathbf{x}\rangle$ be a unit vector; because of Eq. (22), for any i,k and $l=1,2$, denote

$$U(\vec{\theta})_{k,l}^{(i)} = U_1(\vec{\theta}^{(1)})\cdots U_{i-1}(\vec{\theta}^{(i-1)})U_i\left(\vec{\theta}^{(i)}+\Delta_{k,l}^{(i)}\right) U_{i+1}(\vec{\theta}^{(i+1)})\cdots U_n(\vec{\theta}^{(n)}).$$

Then, $\langle\mathbf{x}|\mathbf{w}\rangle = \langle\mathbf{x}|U(\vec{\theta})|0\rangle$ and

$$\frac{\partial}{\partial\theta_k^{(i)}}\langle\mathbf{x}|\mathbf{w}\rangle = \langle\mathbf{x}|\frac{\partial}{\partial\theta_k^{(i)}}U(\vec{\theta})|0\rangle = \frac{1}{2}\langle\mathbf{x}|U(\vec{\theta})_{k,1}^{(i)}|0\rangle - \frac{1}{2}\langle\mathbf{x}|U(\vec{\theta})_{k,2}^{(i)}|0\rangle. \tag{23}$$

Therefore, the gradient vector of E satisfies

$$\frac{\partial E}{\partial\theta_k^{(i)}} = \frac{1}{m}\sum_{t=1}^{m}\psi(\mathbf{x}^{(t)}\cdot\mathbf{w})\|\mathbf{x}^{(t)}\|_2\frac{\partial}{\partial\theta_k^{(i)}}\langle\mathbf{x}^{(t)}|\mathbf{w}\rangle. \tag{24}$$

The updating rule of $\vec{\theta}$ is

$$\theta_k^{(i)}(n+1) = \theta_k^{(i)}(n) - \eta\frac{\partial E}{\partial\theta_k^{(i)}}. \tag{25}$$

With the above notations, we can modify Algorithm 1 into the following algorithm, where $\vec{\theta}$ for $\mathbf{w}(n)$ will be denoted as $\vec{\theta}(n)$. Due to Remark 1, it suffices to consider the case that $\psi(\mathbf{x}^{(t)}\cdot\mathbf{w}(n))\|\mathbf{x}^{(t)}\|_2\frac{\partial}{\partial\theta_k^{(i)}}\langle\mathbf{x}^{(t)}|\mathbf{w}(n)\rangle\geqslant 0$ for all t. By Eq. (23), the gradient is determined by two similar terms. In the following algorithm, we only show how to compute the first term; the other one can be obtained similarly. Also, i,k are fixed indices in the following algorithm.

Algorithm 2. Improved quantum algorithm to estimate the gradient.

Step 1. Prepare $|\phi_1\rangle = \frac{1}{\sqrt{m}}\sum_{t=1}^{m}|t\rangle$.

Step 2. View $|t\rangle$ as a control qubit; we can prepare the following quantum state:

$$|\phi_2\rangle = \frac{1}{\sqrt{m}}\sum_{t=1}^{m}|t\rangle\otimes|\mathbf{x}^{(t)}\rangle\otimes|\mathbf{x}^{(t)}\rangle.$$

Step 3. Apply $I\otimes U^\dagger(\vec{\theta}(n))\otimes U^\dagger(\vec{\theta}(n))_{k,1}^{(i)}$ to $|\phi_2\rangle$ to get

$$|\phi_3\rangle = \frac{1}{\sqrt{m}}\sum_{t=1}^{m}|t\rangle\otimes U^\dagger(\vec{\theta}(n))|\mathbf{x}^{(t)}\rangle\otimes U^\dagger(\vec{\theta}(n))_{k,1}^{(i)}|\mathbf{x}^{(t)}\rangle.$$

Step 4. Apply amplitude estimation, respectively, to estimate the amplitude of $|0\rangle$ of $U(\vec{\theta}(n))^\dagger|\mathbf{x}^{(t)}\rangle$ and of $U^\dagger(\vec{\theta}(n))_{k,1}^{(i)}|\mathbf{x}^{(t)}\rangle$; then we obtain an ϵ-approximate $q_{t,n}$ of $\mathbf{x}^{(t)}\cdot\mathbf{w}(n)$ and an ϵ-approximate $q_{t,i,k}$ of $\|\mathbf{x}^{(t)}\|_2\frac{\partial}{\partial\theta_k^{(i)}}\langle\mathbf{x}^{(t)}|\mathbf{w}\rangle$.

Store them in ancilla registers,

$$|\phi_4\rangle = \frac{1}{\sqrt{m}}\sum_{t=1}^{m}|t\rangle \otimes U^{\dagger}(\vec{\theta}(n))|\mathbf{x}^{(t)}\rangle \otimes U^{\dagger}(\vec{\theta}(n))^{(i)}_{k,1}|\mathbf{x}^{(t)}\rangle \otimes |q_{t,n}\rangle \otimes |q_{t,i,k}\rangle.$$

Step 5. Apply control rotation to $|\phi_4\rangle$ to get

$$|\phi_5\rangle = \frac{1}{\sqrt{m}}\sum_{t=1}^{m}|t\rangle \otimes U(\vec{\theta}(n))^{\dagger}|\mathbf{x}^{(t)}\rangle \otimes U^{\dagger}(\vec{\theta}(n))^{(i)}_{k,1}|\mathbf{x}^{(t)}\rangle \otimes |q_{t,n}\rangle \otimes |q_{t,i,k}\rangle \otimes [\sqrt{\psi(q_{t,n})q_{t,i,k}L}|0\rangle + \sqrt{1-|\psi(q_{t,n})q_{t,i,k}|L}|1\rangle],$$

where $L = 1/\max_t |\psi(q_{t,n})\|\mathbf{x}^{(t)}\|_2|$.

Step 6. Undo steps 2–4 to get

$$|\phi_6\rangle = \frac{\sqrt{L}}{\sqrt{m}}\sum_{t=1}^{m}\sqrt{\psi(q_{t,n})q_{t,i,k}}|t\rangle|0\rangle + (\cdots)|1\rangle.$$

Step 7. Apply amplitude estimation to estimate the amplitude of $|0\rangle$ in $|\phi_6\rangle$.

Similar to the proof of Theorem 2, we have the following:

Theorem 3. Assume that the oracle (9) exists for the training samples. Then, Algorithm 2 returns an ϵ-approximate of $\partial E/\partial\theta_k^{(i)}$ in time,

$$O\{(\log_2 m)(\max_t \|\mathbf{x}^{(t)}\|_2)[\max_t |\psi(\mathbf{x}^{(t)}\cdot\mathbf{w}(n))|]/\epsilon^2\}. \quad (26)$$

Remark 2. In step 5, the best choice of the parameter L should be $1/\max_t |\psi(q_{t,n})q_{t,i,k}|$. However, $|\frac{\partial}{\partial\theta_k^{(i)}}\langle\mathbf{x}^{(t)}|\mathbf{w}\rangle| \leqslant 1$; thus we can simply set $L = 1/\max_t |\psi(q_{t,n})\|\mathbf{x}^{(t)}\|_2|$. This actually simplifies the choice of L, another advantage of Algorithm 2.

The unitary $U(\vec{\theta})$ provides an easy approach to estimate the inner product $\langle\mathbf{x}|\mathbf{w}\rangle$ in Algorithm 2. The new representation (i.e., Theorem 1) of real vectors also provides us with a better method to implement Algorithm 2 in a quantum computer. This will be discussed in more detail in the next section.

V. CIRCUIT IMPLEMENTATION

In this section, we draw the quantum circuit to run Algorithm 2. Instead of drawing the whole quantum circuit, we only show its nontrivial parts. This includes the following: (i) The circuit to implement the amplitude estimation (step 4 and step 7). (ii) The circuit to implement the control rotation (step 5).

The amplitude estimation algorithm is an application of the quantum phase estimation (QPE) (see the Appendix). In a quantum computer, QPE has an efficient circuit [26]; see Fig. 4 below.

By Fig. 4, to implement amplitude estimation in the quantum circuit, it suffices to implement G in the quantum circuit. By Eq. (A1), this further reduces to build a quantum circuit to implement U, the unitary operator to prepare the given quantum state.

In Theorem 1, denote the angle parameter for $\mathbf{x}^{(t)}$ as $\vec{\theta}(t)$; then, $|\mathbf{x}^{(t)}\rangle = U(\vec{\theta}(t))|0\rangle^{\otimes\log_2 d}$. In step 4 of Algorithm 2, we apply the amplitude estimation algorithm two times to estimate the amplitude of $|0\rangle$. For the first time, $U = U^{\dagger}(\vec{\theta}(n))U(\vec{\theta}(t))$; for the second time,

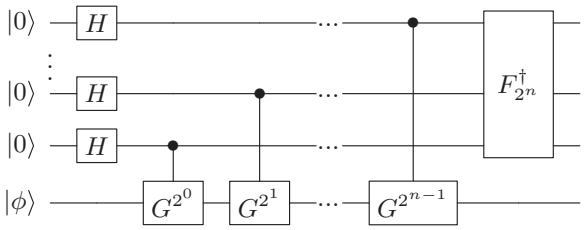

FIG. 4. Circuit to implement QPE, where H is Hadamard gate, F_{2^n} is quantum Fourier transform, and G is the unitary given in Eq. (A1).

$U = U^{\dagger}(\vec{\theta}(n))^{(i)}_{k,l}U(\vec{\theta}(t))$. Both of them have efficient circuit implementations due to Eqs. (6) and (22) and the circuit (see Fig. 3) to run each $U_i(\vec{\theta}^{(i)})$.

To give the quantum circuit of step 4, we consider the implementation of Eq. (A3). Before that, we need the following simple fact [22]: If $f(x) = ax + b$, then there is an efficient quantum circuit to implement U_f. Therefore, any polynomial f with $O(1)$ degree has an efficient circuit to implement U_f. As a result, we have the following:

Proposition 1. If $f(x) = \cos x$ or $\arccos(x)$, then there is an efficient quantum circuit to implement U_f.

Proof. By Taylor series, we can approximate $\cos x$ to high precision by a low degree (e.g., degree 10) polynomial when $0 \leqslant x \leqslant \pi$. The is also true for $\arccos(x)$. ■

There may be another better method to implement the cosine and arccosine functions in a quantum computer. However, the above result is already sufficient for us. Certainly, we need to truncate the Taylor series to a certain order to make sure the polynomial gives a high-precision approximate of the cosine or arccosine function. Therefore, by Fig. 4 and Proposition 1, there is an efficient circuit to implement the algorithm to obtain (A3) when $f(x)$ is a low degree polynomial of $\cos(x)$. This gives the quantum circuit to implement step 4 of Algorithm 2 since $f(x) = \|\mathbf{x}^t\|_2\|\mathbf{w}(n)\|_2[1-2(\cos x)^2]$, which is discussed below Algorithm 1.

Next, we give an analysis of the circuit to implement the control rotation. For simplicity, we reexpress the control rotation in step 5 in the following, more general, form:

$$\textstyle\sum_i a_i|i\rangle|v_i\rangle|0\rangle \mapsto \sum_i a_i|i\rangle|v_i\rangle \otimes R(\theta_i)|0\rangle, \quad (27)$$

where $\cos\theta_i = f(v_i)$. Thus, $\theta_i = \arccos[f(v_i)]$, and it is unique when restricted to the interval $[0,\pi]$. To implement (27) in a quantum circuit, we decompose it into the following five steps:

$$\begin{aligned}
&\textstyle\sum_i a_i|i\rangle|v_i\rangle|0,0,0\rangle \\
\xrightarrow{I\otimes U_f}\ &\textstyle\sum_i a_i|i\rangle|v_i\rangle|f(v_i))\rangle|0,0\rangle \\
\xrightarrow{I\otimes I\otimes U_{\arccos}}\ &\textstyle\sum_i a_i|i\rangle|v_i\rangle|f(v_i))|\theta_i\rangle|0\rangle \\
\rightarrow\ &\textstyle\sum_i a_i|i\rangle|v_i\rangle|f(v_i))|\theta_i\rangle \otimes R(\theta_i)|0\rangle \\
\xrightarrow{I\otimes I\otimes U^{-1}_{\arccos}}\ &\textstyle\sum_i a_i|i\rangle|v_i\rangle|f(v_i))|0\rangle \otimes R(\theta_i)|0\rangle \\
\xrightarrow{I\otimes U_f^{-1}}\ &\textstyle\sum_i a_i|i\rangle|v_i\rangle|0,0\rangle \otimes R(\theta_i)|0\rangle.
\end{aligned} \quad (28)$$

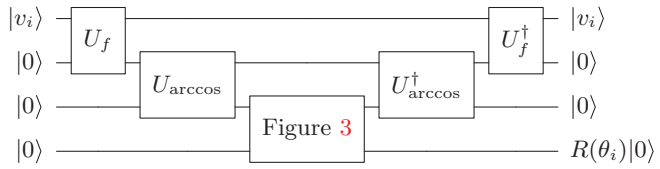

FIG. 5. Circuit to implement control rotation.

In Proposition 1, we use a constant degree polynomial to approximate the arccosine function to high precision; thus we can approximate $\theta_i = \arccos[f(v_i)]$ to high precision in the third line of Eq. (28). Also because of Proposition 1, $U_{\arccos}$ is efficient. Therefore, when U_f is efficient, the main circuit we need to construct is the fourth row of Eq. (28), which is already depicted in Fig. 3. At last, we give the quantum circuit to implement procedure (27) in Fig. 5 below.

VI. APPLICATION: SOLVING CONVEX OPTIMIZATIONS

Convex optimization studies the problem of minimizing convex functions over convex sets [27]. It has wide applications in control systems, signal processing, communications and networks, electronic circuit design, data analysis and modeling, etc. In this section, we study the quantum algorithm to solving the convex optimization problem based on an interior-point method, called the barrier method. The main idea is to change the convex optimization problem into a similar form to the minimization problem (1) so that Algorithm 2 can be applied by a simple modification. Therefore, in this section, we only cover the reduction from the convex optimization problem to the minimization problem (1). The corresponding quantum algorithms will not be restated.

The general convex optimization problem has the following form:

$$\begin{aligned} &\min_{\mathbf{w}} && f_0(\mathbf{w}), \\ &\text{subject to} && f_i(\mathbf{w}) \leqslant 0,\ i = 1, 2, \ldots, m, \\ &&& A\mathbf{w} = \mathbf{b}, \end{aligned} \tag{29}$$

where $f_0, f_1, \ldots, f_m : \mathbb{R}^d \to \mathbb{R}$ are convex and twice continuously differentiable, and $A \in \mathbb{R}^{p\times d}$ has $\operatorname{rank}(A) = p < d$. Assume that the problem is solvable. The optimal solution of (29) will be denoted as $\mathbf{w}^\star$.

Interior-point methods are a certain class of algorithms that solve convex optimizations. One particular type of interior-point method is called the barrier method. It was first proposed Fiacco and McCormick in the 1960s [28]. A central idea to solve (29) by the barrier method is approximately formulating the inequality constrained problem (29) as an equality constrained problem to which the typical optimization method, such as the gradient descent method or Newton's method, can be applied.

Let $I_- : \mathbb{R} \to \mathbb{R}$ be the following indicator function for the nonpositive real numbers:

$$I_-(y) = \begin{cases} 0, & y \leqslant 0 \\ +\infty, & y > 0. \end{cases}$$

Then, Eq. (29) is equivalent to the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{w}} && f_0(\mathbf{w}) + \textstyle\sum_{i=1}^m I_-(f_i(\mathbf{w})), \\ &\text{subject to} && A\mathbf{w} = \mathbf{b}. \end{aligned} \tag{30}$$

The function $I_-(y)$ is not continuous; however, we can approximate it by a differentiable function,

$$\widehat{I}_-(y) := -t^{-1}\log_2(-y),$$

with domain $\mathbb{R}^{<0}$, where $t > 0$ is a parameter that sets the accuracy of the approximation. As t increases, the approximation becomes more accurate.

Substituting $\widehat{I}_-$ for I_- in (30) gives an approximation of the convex optimization problem (29) as follows:

$$\begin{aligned} &\min_{\mathbf{w}} && f_0(\mathbf{w}) + \textstyle\sum_{i=1}^m \widehat{I}_-[f_i(\mathbf{w})] \\ &&= \ & f_0(\mathbf{w}) - \textstyle\sum_{i=1}^m t^{-1}\log_2[-f_i(\mathbf{w})], \\ &\text{subject to} && A\mathbf{w} = \mathbf{b}. \end{aligned} \tag{31}$$

It has been shown in [27] that the error between the optimal value of (29) and (31) is bounded by m/t. This implies that (31) is a good approximate of (29) if we choose $t = m/\epsilon$. However, choosing $t = m/\epsilon$ directly does not work very well in many cases. As a result, this choice is rarely used in practice. A simple extension is solving a sequence of unconstrained minimization problems in the form (31), using the last point found as the starting point for the next unconstrained minimization problem. In other words, we compute the optimal solution $\mathbf{w}^\star(t)$ of (31) for a sequence of increasing values of t, until $t \geqslant m/\epsilon$. With this idea, a simple version of the barrier method can be stated as follows:

Algorithm 3. Barrier method [28].

Input. Strictly feasible $\mathbf{w}$, an initial choice of $t > 0$, $\mu > 1$, and a tolerance $\epsilon > 0$.

Repeat.

(a) Centering step: compute $\mathbf{w}^\star(t)$ by minimizing (31) with initial value $\mathbf{w}$.

(b) Update: $\mathbf{w} = \mathbf{w}^\star(t)$.

(c) Stopping criterion: stop if $m/t < \epsilon$.

(d) Increase: $t = \mu t$.

In the above algorithm, strictly feasible means $\mathbf{w}$ satisfies $A\mathbf{w} = \mathbf{b}$ and $f_i(\mathbf{w}) < 0$ for all $i = 1, \ldots, m$. Besides the choice of the initial parameters, the critical point of Algorithm 3 is the iteration methods used in step 1 to solve (31) for a fixed t. Therefore, Algorithm 2 can play an active role in improving the efficiency of this step. With a similar structure to Algorithm 2, we can improve the dependence of the complexity of classical algorithm on m to $\log_2 m$. Note that here m refers to the number of constraints in the convex optimization problem.

To apply Algorithm 2, one problem we need to solve is the calculation of $f_i(\mathbf{w})$. We can either design a quantum algorithm to calculate it or just use an oracle to query the result of the classical computation. In the first case, we may obtain further speedup over the classical algorithm. However, the designing of such a quantum algorithm may not be easy. It depends on the specific expression of f_i. When f_i only relates to the inner product [see Eq. (20)], then by the swap test or amplitude estimation technique, constructing such a quantum

algorithm is not difficult. In the following, we show some examples that we can use Algorithm 2 directly.

Example 1. Linear programming. The inequality form of linear programming can be stated as follows:

$$\begin{aligned} &\min_{\mathbf{w}} && \mathbf{c}\cdot\mathbf{w}, \\ &\text{subject to} && A\mathbf{w}\leqslant\mathbf{b}, \end{aligned}$$

where $A\in\mathbb{R}^{m\times d}$. Denote the ith row of A as $\mathbf{a}^{(i)}$; then, by the above analysis, it suffices to apply Algorithm 2 to solve the following problem:

$$\min_{\mathbf{w}} \mathbf{c}\cdot\mathbf{w}-\sum_{i=1}^{m} t^{-1}\log_2(b_i-\mathbf{a}^{(i)}\cdot\mathbf{w}).$$

Example 2. Quadratic programming. A quadratic programming with no equality constraints can be expressed in the form

$$\begin{aligned} &\min_{\mathbf{w}} && \tfrac{1}{2}\mathbf{w}^T P\mathbf{w}+\mathbf{q}\cdot\mathbf{w}+\mathbf{c}, \\ &\text{subject to} && A\mathbf{w}\leqslant\mathbf{b}, \end{aligned}$$

where $P\in\mathbb{R}^{d\times d}$ is a symmetric matrix and $A\in\mathbb{R}^{m\times d}$. It contains linear programming as a special case when $P=0$. Similarly, it suffices to apply Algorithm 2 to solve

$$\min_{\mathbf{w}} \tfrac{1}{2}\mathbf{w}^T P\mathbf{w}+\mathbf{q}\cdot\mathbf{w}+\mathbf{c}-\sum_{i=1}^{m} t^{-1}\log_2(b_i-\mathbf{a}^{(i)}\cdot\mathbf{w}).$$

Example 3. Geometric programming. The geometric programming [29] considers that the optimization problem with objective and constraint functions are posynomials. A posynomial is a positive linear summation of monomial function in the form $x_1^{a_1}\cdots x_d^{a_d}$, where $a_1,\ldots,a_d\in\mathbb{R}$ and $x_1,\ldots,x_d\in\mathbb{R}^{>0}$. It can be transformed into convex optimization by setting $x_i=\exp(w_i)$ for some new variables $w_1,\ldots,w_d$. If we further set $\mathbf{a}=(a_1,\ldots,a_d)$ and $\mathbf{w}=(w_1,\ldots,w_d)$, then a term of posynomial has the form $\exp(\mathbf{a}\cdot\mathbf{w}+b)$, where $\exp(b)$ refers to the positive coefficient. Now we can transform the geometric programming into a convex optimization by taking the logarithm of the objective and constraint functions. Therefore, an unconstrained geometric programming has the following form:

$$\min_{\mathbf{w}} \log_2\left[\sum_{i=1}^{m}\exp(\mathbf{a}^{(i)}\cdot\mathbf{w}+b_i)\right],$$

where $\mathbf{a}^{(i)},\mathbf{w}\in\mathbb{R}^d$. In this problem, we do not need to apply the barrier method. We can apply Algorithm 2 to achieve speedup at m directly.

VII. CONCLUSIONS

In this paper, we showed how to apply a variational quantum algorithm to improve the training of neural networks. As an application, we indicated how to applied similar algorithms to solve convex optimizations. However, the improvement may not be optimistic for convex optimization since the number of unknown variables is also a significant factor that has not being considered seriously in this paper. Due to the variational algorithm, we can use few parameters to describe the unknown vector. This may provide further convenience to run the gradient method in a quantum computer. We leave this as a problem for future research.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China Project No. 11671388 and the CAS Frontier Key Project No. QYZDJ-SSW-SYS022.

APPENDIX: AMPLITUDE ESTIMATION

In this section, we briefly describe the amplitude estimation algorithm [21]. It is an important technique that will be used in this paper.

Let $|\phi\rangle=U|0\rangle^{\otimes k}=\sin\theta|0\rangle|u\rangle+\cos\theta|1\rangle|v\rangle$ be a quantum state that can be prepared by a unitary operator U in time $O(T)$. Amplitude estimation is a quantum algorithm that can estimate $\sin^2\theta$ and $\cos^2\theta$. The basic idea can be described as follows.

Let Z be the two-dimensional Pauli-Z matrix that maps $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $-|1\rangle$. Denote

$$\begin{aligned} G&=(I-2|\phi\rangle\langle\phi|)(Z\otimes I)\\ &=U(I-2|0\rangle^{\otimes k}\langle 0|^{\otimes k})U^\dagger(Z\otimes I). \end{aligned} \tag{A1}$$

We can check that

$$G=\begin{pmatrix}\cos 2\theta & \sin 2\theta\\ -\sin 2\theta & \cos 2\theta\end{pmatrix}$$

in the space spanned by $\{|0\rangle|u\rangle,|1\rangle|v\rangle\}$. The eigenvalues of G are $e^{\pm\mathbf{i}2\theta}$ and the corresponding eigenvectors are $|w_\pm\rangle=\frac{1}{\sqrt{2}}(|0\rangle|u\rangle\pm\mathbf{i}|1\rangle|v\rangle)$, respectively.

To apply the quantum phase estimation (QPE) to estimate θ, we rewrite $|\phi\rangle$ as $|\phi\rangle=-\frac{\mathbf{i}}{\sqrt{2}}(e^{\mathbf{i}\theta}|w_+\rangle-e^{-\mathbf{i}\theta}|w_-\rangle)$. We perform the QPE on G with the initial state $|0\rangle^n|\phi\rangle$ for some $n=O(\log_2 1/\delta\epsilon)$. Then, with probability at least $1-\delta$, we get an approximate of the following state:

$$-\frac{\mathbf{i}}{\sqrt{2}}(e^{\mathbf{i}\theta}|y\rangle|w_1\rangle-e^{-\mathbf{i}\theta}|-y\rangle|w_2\rangle), \tag{A2}$$

where $y\in\mathbb{Z}_{2^n}$ satisfies $|\theta-y\pi/2^n|\leqslant\epsilon$. The time complexity of the above procedure is $O(T/\epsilon\delta)$. By performing measurements on (A2), we will obtain ϵ-approximates of $\pm\theta$, from which ϵ-approximates of $\sin^2\theta$ and $\cos^2\theta$ can be obtained.

To further apply the information of θ, we do not need to perform measurements on (A2). Now let $f(y)$ be an even function, i.e., $f(y)=f(-y)$. Assume that there is an oracle U_f to implement f, that is, $U_f|x,y\rangle=|x,y\oplus f(x)\rangle$ is efficient. Then, by (A2), we can get

$$|\phi\rangle|f(\tilde{\theta})\rangle, \tag{A3}$$

by adding a register to store $f(\tilde{\theta})$ and undoing QPE, where $|\theta-\tilde{\theta}|\leqslant\epsilon$.

To apply the amplitude estimation algorithm to establish the quantum algorithm in this paper, we consider the following application of amplitude estimation. Let $\sum_{j=1}^{K}\alpha_j|j\rangle$ be a

quantum state that can be prepared in time $O(T_1)$. Assume that $|\phi_j\rangle = \sin\theta_j|0\rangle|u_j\rangle + \cos\theta_j|1\rangle|v_j\rangle$ can be prepared in time $O(T_2)$ for all j. Let f be a function such that $|f(x)| \leqslant 1$ and U_f is efficient. The problem we consider is how to prepare

$$\sum_{j=1}^{K} \alpha_j f(\cos\tilde{\theta}_j)|j\rangle|0\rangle + |0^{\perp}\rangle \tag{A4}$$

efficiently, where $|\theta_j - \tilde{\theta}_j| \leqslant \epsilon$ and $|0^{\perp}\rangle$ indicates some state that is orthogonal to the first part.

First, by viewing $|j\rangle$ as a control qubit, we can prepare $\sum_{j=1}^{K} \alpha_j|j\rangle|\phi_j\rangle$. Since the cosine function is even, we can obtain

$$\sum_{j=1}^{K} \alpha_j|j\rangle|\phi_j\rangle|f(\cos\tilde{\theta}_j)\rangle \tag{A5}$$

by performing (A3) conditionally, that is, viewing $|j\rangle$ as a control qubit to perform the amplitude estimation in parallel.

Then, apply control rotation (similar to the Harrow-Hassidim-Lloyd (HHL) algorithm [30]) to the quantum state (A5) to obtain

$$\sum_{j=1}^{K} \alpha_j|j\rangle|\phi_j\rangle|f(\cos\tilde{\theta}_j)\rangle \otimes \left[f(\cos\tilde{\theta}_j)|0\rangle + \sqrt{1 - f^2(\cos\tilde{\theta}_j)}|1\rangle\right]. \tag{A6}$$

Finally, undo the amplitude estimation and unprepare the state $\sin\theta_j|0\rangle|u_j\rangle + \cos\theta_j|1\rangle|v_j\rangle$; then we get

$$\sum_{j=1}^{K} \alpha_j|j\rangle[f(\cos\tilde{\theta}_j)|0\rangle + \sqrt{1 - f^2(\cos\tilde{\theta}_j)}|1\rangle]. \tag{A7}$$

This gives the desired state (A4).

In the above procedure, we apply the control operation several times. Usually, the complexity of the control operation should be linear at K. However, if we have a uniform quantum circuit that can prepare $|\phi_j\rangle$ for all j, the complexity of the control amplitude estimation is also independent of K. For instance, this is achievable if we have a unitary operator U such that $U|j\rangle|0\rangle = |j\rangle|\phi_j\rangle$ for all j. In this case, the total complexity of the above procedure will be $O(T_1 + T_2/\epsilon)$. Actually, the data structure proposed in Sec. II can be used to build such a unitary operator.

[1] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed. (MIT Press, Cambridge, 2014).

[2] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. (Pearson, London, 2009).

[3] Y. Bengio, Y. LeCun, and G. Hinton, Nature (London) **521**, 436 (2015).

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016).

[5] J. Schmidhuber, Neural Networks **61**, 85 (2015).

[6] J. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New J. Phys. **18**, 023023 (2016).

[7] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, Quantum Sci. Technol. **3**, 030502 (2018).

[8] E. Farhi, J. Goldstone, and S. Gutmann, arXiv:1411.4028.

[9] G. G. Guerreschi and M. Smelyanskiy, arXiv:1701.01450v1.

[10] A. Peruzzo, J. McClean, P. Shadbolt, M. Yung, X. Zhou, P. Love, A. Aspuru-Guzik, and J. Obrien, Nat. Commun. **5**, 4213 (2014).

[11] D. Wecker, M. B. Hastings, and M. Troyer, Phys. Rev. A **92**, 042303 (2015).

[12] E. Farhi and H. Neven, arXiv:1802.06002v2.

[13] G. Verdon, M. Broughton, and J. Biamonte, arXiv:1712.05304.

[14] Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost, and P. Wittek, arXiv:1806.11463.

[15] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Phys. Rev. A **98**, 032309 (2018).

[16] M. Schuld, A. Bocharov, K. Svore, and N. Wiebe, arXiv:1804.00633 .

[17] M. Schuld and N. Killoran, Phys. Rev. Lett. **122**, 040504 (2019).

[18] I. Kerenidis and A. Prakash, in *8th Innovations in Theoretical Computer Science Conference*, Leibniz International Proceedings in Informatics, Vol. 67, edited by C. H. Papadimitriou (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017), pp. 49:1–49:21.

[19] V. Giovannetti, S. Lloyd, and L. Maccone, Phys. Rev. Lett. **100**, 160501 (2008).

[20] L. Grover and T. Rudolph, arXiv:quant-ph/0208112.

[21] G. Brassard, P. Høyer, and M. Mosca, Quantum Inf. Comput. **305**, 53 (2002).

[22] E. Rieffel and W. Polak, *Quantum Computing - A Gentle Introduction* (MIT Press, Cambridge, 2011).

[23] It costs $O(d)$ to estimate the inner product $\mathbf{x}^t \cdot \mathbf{w}$. Each term of $\mathbf{g}(\mathbf{w})$ contains m summations, and $\mathbf{g}(\mathbf{w})$ contains d terms. Totally, the complexity of each step of the iteration is $O(d^2m)$.

[24] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, Phys. Rev. Lett. **87**, 167902 (2001).

[25] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, Phys. Rev. A **99**, 032331 (2019).

[26] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).

[27] S. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, 2004).

[28] A. Fiacco and G. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* (Society for Industrial and Applied Mathematics, Philadelphia, 1968).

[29] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi, Optimization Engineer. **8**, 67 (2007).

[30] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).